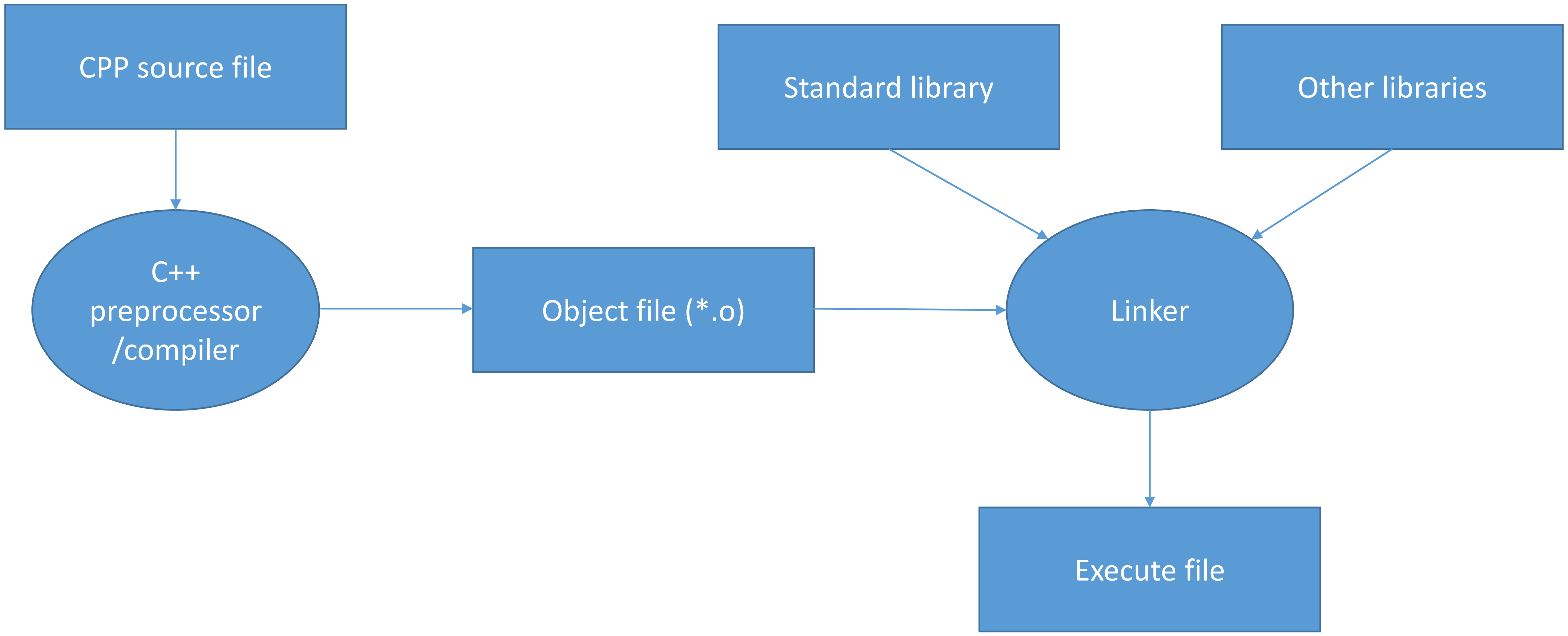
C++ Compile process and compiler tools

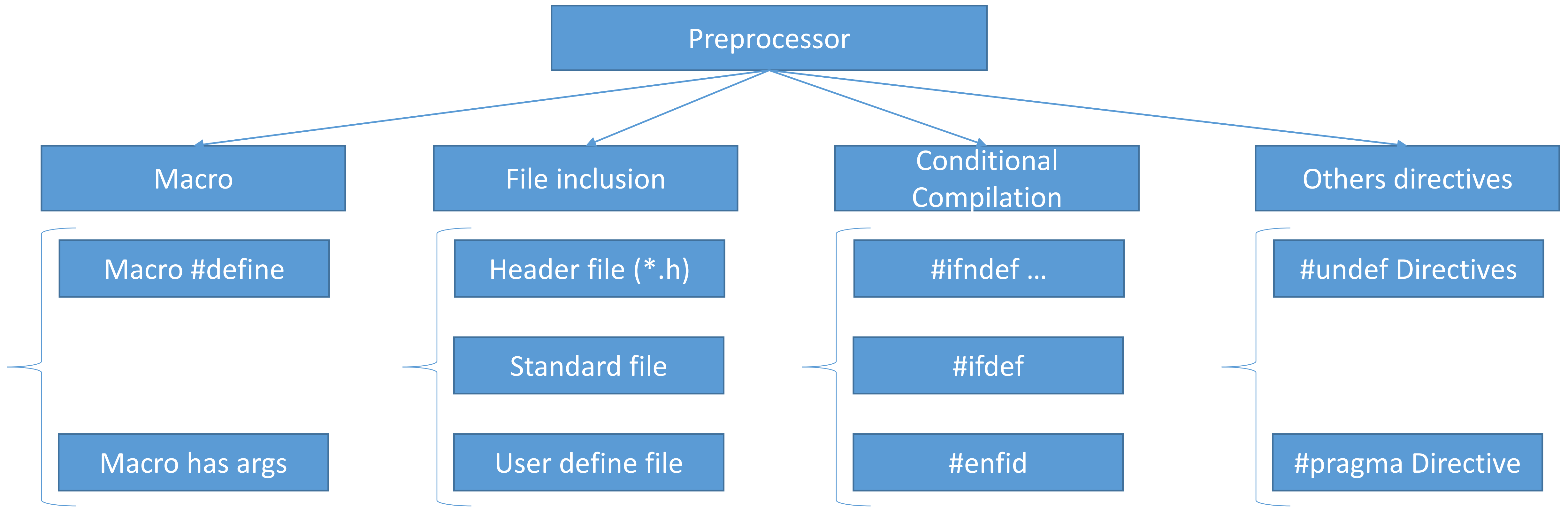
Nguyen Ngoc Hung

Content

C++/C compilation Process



C++/C compilation Process

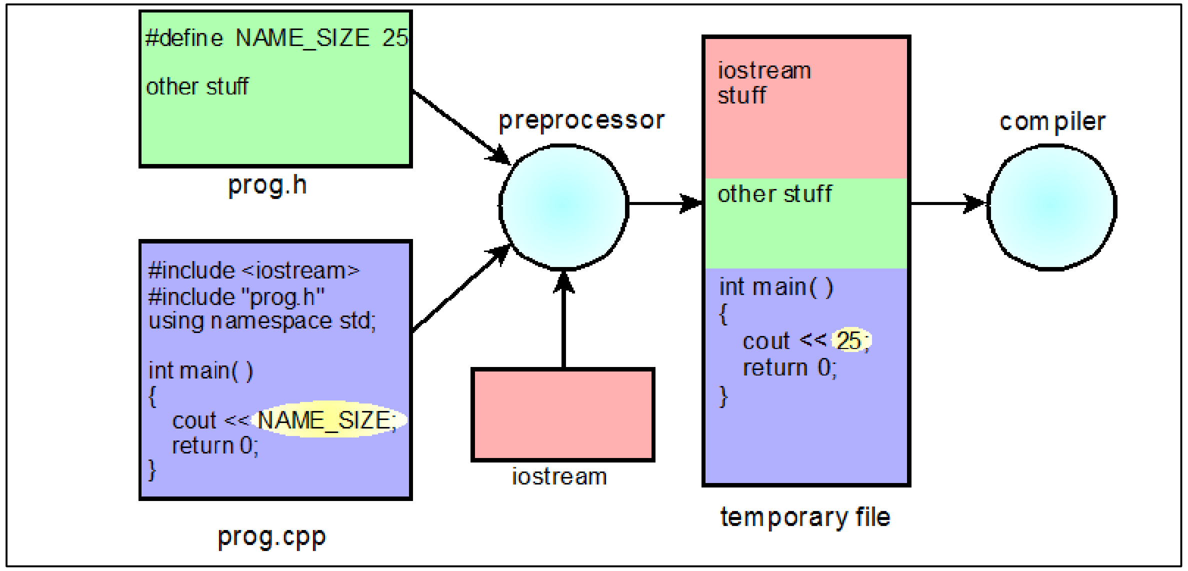


C++/C compilation Process

```
#include <stdio.h>         -> Inclusion file header file.
#include <iostream>        -> Standard file
#include “myfile.h”        -> user define file

#define LIMIT 5            -> macro #define
#define sum(a,b) (a+b)     -> macro with arguments

#ifdef __IOSTREAM_H        -> conditional compilation
#undef __IOSTREAM_H        -> undefined conditional compilation
#endif


int main()
{
    for (int i = 0; i <  LIMIT; i++)
        {
       printf("%d\n"i);
        }
    return 0;
}
```

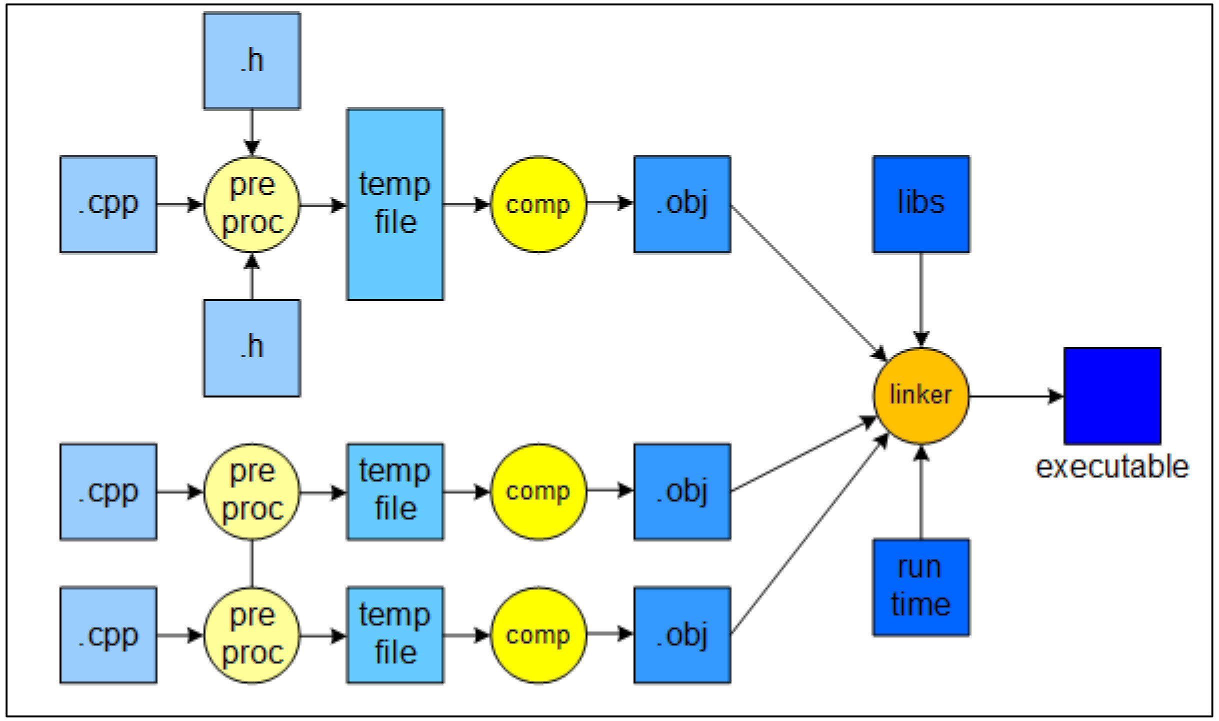
C++/C compilation Process

Preprocessor



The preprocessor handles directives that begin with the # character and creates a temporary file to store its output. The compiler reads the temporary file and continues the compilation process.

Compilation



C++/C compilation Process

- **Some popular compiler**

Compiler	Author	Window os	Unix-like	C++ version
Embarcadero	Embarcadero	YES	IOS, Android	C89/C99
GCC C/g++	GNU Project	Yes	Yes	Up to C18
Microsoft Visual C++	Microsoft	Yes	No	Up to C11
AMD Optimizing C/C++ Compiler (AOCC)	AMD	No	Yes	

Compiler by GNU Compiler Collection

- GCC options (1)

Overall option	C language option	C++ language option	Objective C/C++ language option	Diagnostic message format option	Warning option
C and C only warning option	Debugging option	Optimization option	Program instrument option	Preprocessor option	Assembler option
Linker option	Directory option	Code Generation option	Developer option	Machine Dependent option	

(1) Ref: https://gcc.gnu.org/onlinedocs/gcc-6.1.0/gcc.pdf

Compiler by GNU Compiler Collection

- **Overall option**

-X – c –S –E – o file – v --help.

-X: determined exactly language.

-c: compile or assemble the source file but not link -> file with *.o

-S: Stop after the stage of compilation proper do not assemble. -> file with *.s

-E: Stop after the stage of preprocessing, the output file is in form of preprocessing of source code.

-o file: g++ -o + output file + source code file. Output can be a executive file.

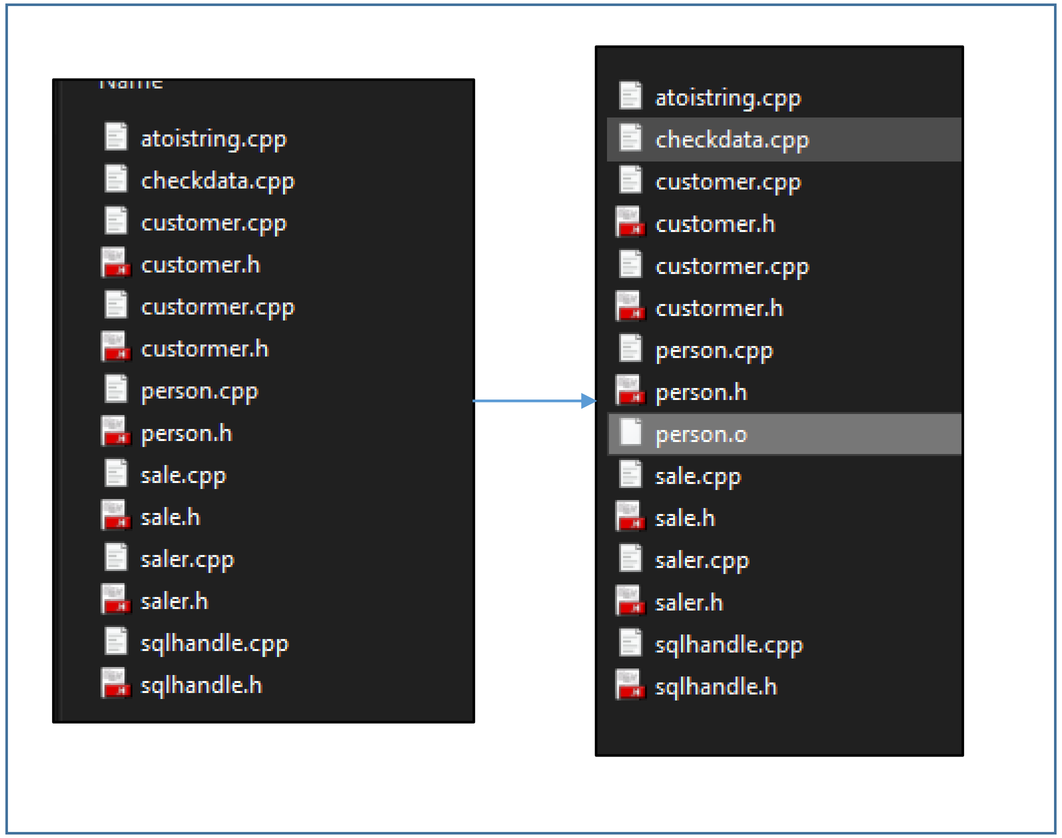
-v: print (on standard output) the commands executed to run at the stage of compilation.

Compiler by GNU Compiler Collection

- **Overall option**

Example: compile person.cpp file by using –X and –c, o:

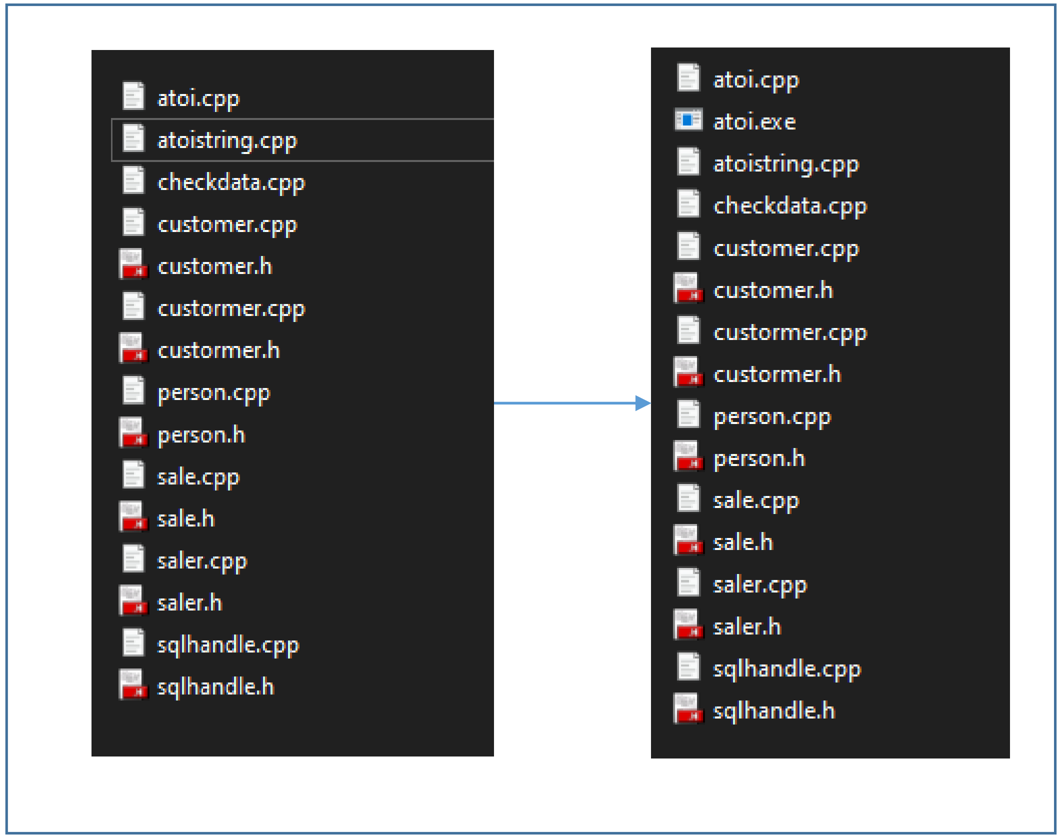
g++ -X –cpp –c person.cpp



```
atoistring.cpp
checkdata.cpp
customer.cpp
customer.h
custormer.cpp
custormer.h
person.cpp
person.h
sale.cpp
sale.h
saler.cpp
saler.h
sqlhandle.cpp
sqlhandle.h
```

```
atoistring.cpp
checkdata.cpp
customer.cpp
customer.h
custormer.cpp
custormer.h
person.cpp
person.h
person.o
sale.cpp
sale.h
saler.cpp
saler.h
sqlhandle.cpp
sqlhandle.h
```

g++ -X -cpp -o atoi.exe atoi.cpp



```
atoi.cpp
atoistring.cpp
checkdata.cpp
customer.cpp
customer.h
custormer.cpp
custormer.h
person.cpp
person.h
sale.cpp
sale.h
saler.cpp
saler.h
sqlhandle.cpp
sqlhandle.h
```

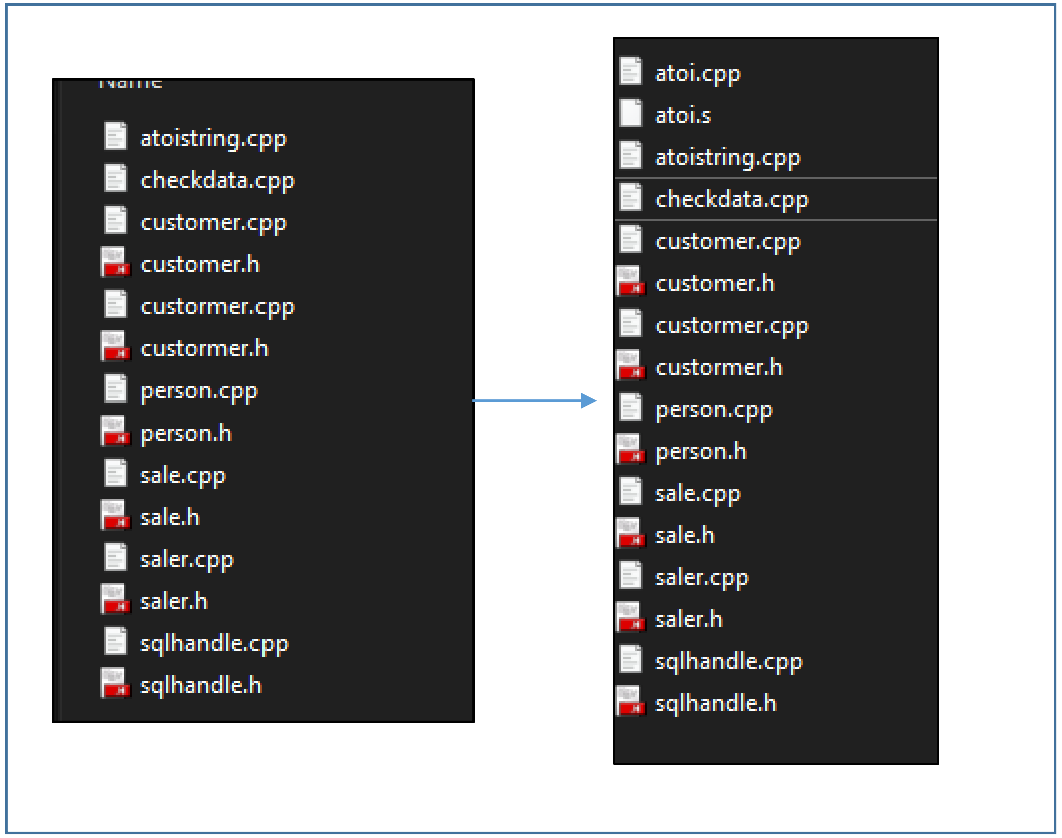
```
atoi.cpp
atoi.exe
atoistring.cpp
checkdata.cpp
customer.cpp
customer.h
custormer.cpp
custormer.h
person.cpp
person.h
sale.cpp
sale.h
saler.cpp
saler.h
sqlhandle.cpp
sqlhandle.h
```

Compiler by GNU Compiler Collection

- **Overall option**

Example: using –S

g++ -S atoi.cpp



```
Name
atoistring.cpp
checkdata.cpp
customer.cpp
customer.h
custormer.cpp
custormer.h
person.cpp
person.h
sale.cpp
sale.h
saler.cpp
saler.h
sqlhandle.cpp
sqlhandle.h
```

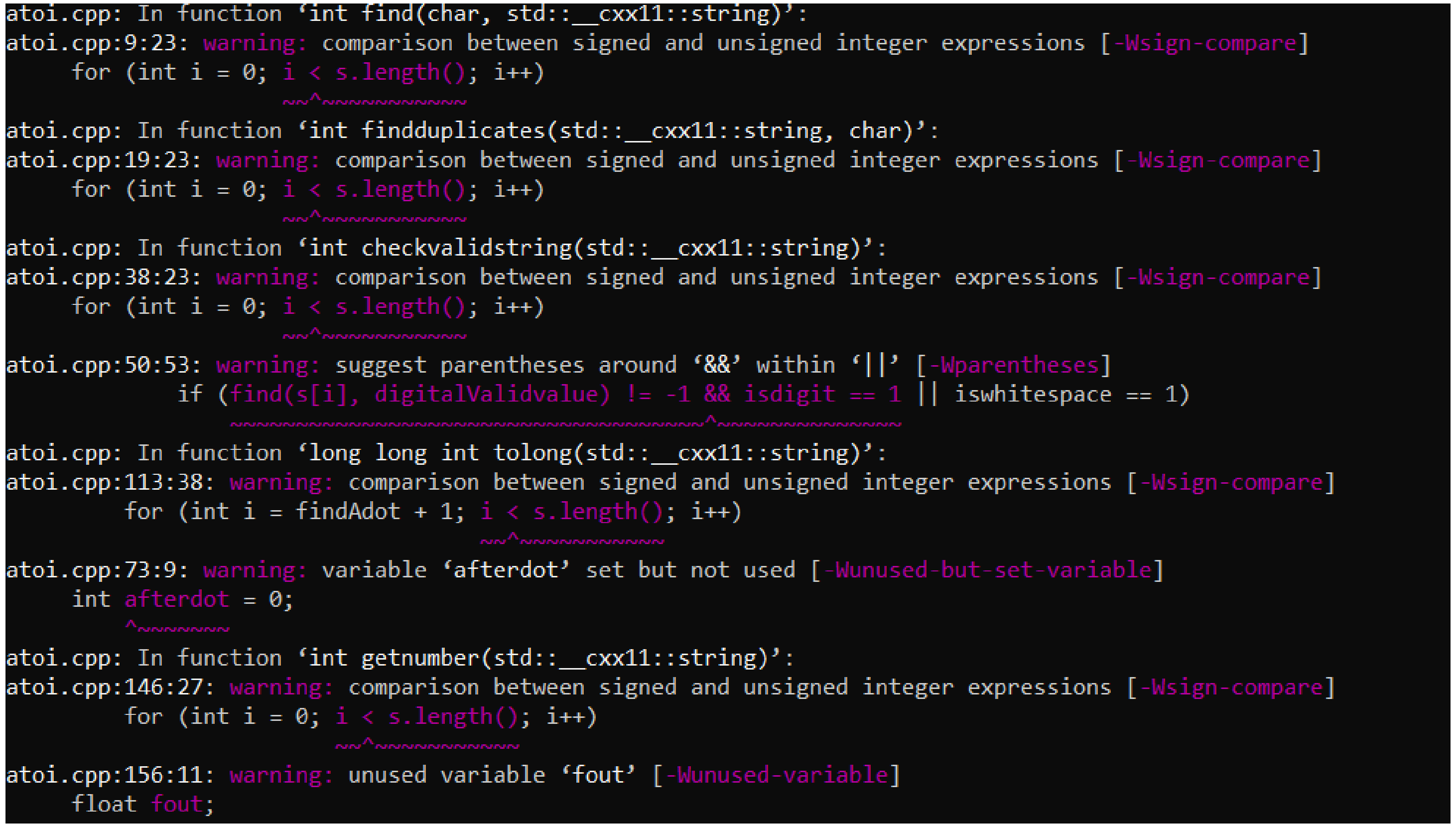
```
atoi.cpp
atoi.s
atoistring.cpp
checkdata.cpp
customer.cpp
customer.h
custormer.cpp
custormer.h
person.cpp
person.h
sale.cpp
sale.h
saler.cpp
saler.h
sqlhandle.cpp
sqlhandle.h
```

Compiler by GNU Compiler Collection

- **Some more example:**

+: Compile with gbd and warning: g++/gcc –g –Wall inputfile.cpp/c –o ouputfile.o/exe

g++ -g -Wall atoi.cpp -o atoi.exe



```
atoi.cpp: In function ‘int find(char, std::__cxx11::string)’:
atoi.cpp:9:23: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
     for (int i = 0; i < s.length(); i++)
                     ~~^~~~~~~~~~~~
atoi.cpp: In function ‘int findduplicates(std::__cxx11::string, char)’:
atoi.cpp:19:23: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
     for (int i = 0; i < s.length(); i++)
                     ~~^~~~~~~~~~~~
atoi.cpp: In function ‘int checkvalidstring(std::__cxx11::string)’:
atoi.cpp:38:23: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
     for (int i = 0; i < s.length(); i++)
                     ~~^~~~~~~~~~~~
atoi.cpp:50:53: warning: suggest parentheses around ‘&&’ within ‘||’ [-Wparentheses]
             if (find(s[i], digitalValidvalue) != -1 && isdigit == 1 || iswhitespace == 1)
                 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^~~~~~~~~~~~~~
atoi.cpp: In function ‘long long int tolong(std::__cxx11::string)’:
atoi.cpp:113:38: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
         for (int i = findAdot + 1; i < s.length(); i++)
                                    ~~^~~~~~~~~~~~
atoi.cpp:73:9: warning: variable ‘afterdot’ set but not used [-Wunused-but-set-variable]
     int afterdot = 0;
         ^~~~~~~~
atoi.cpp: In function ‘int getnumber(std::__cxx11::string)’:
atoi.cpp:146:27: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
         for (int i = 0; i < s.length(); i++)
                         ~~^~~~~~~~~~~~
atoi.cpp:156:11: warning: unused variable ‘fout’ [-Wunused-variable]
     float fout;
```

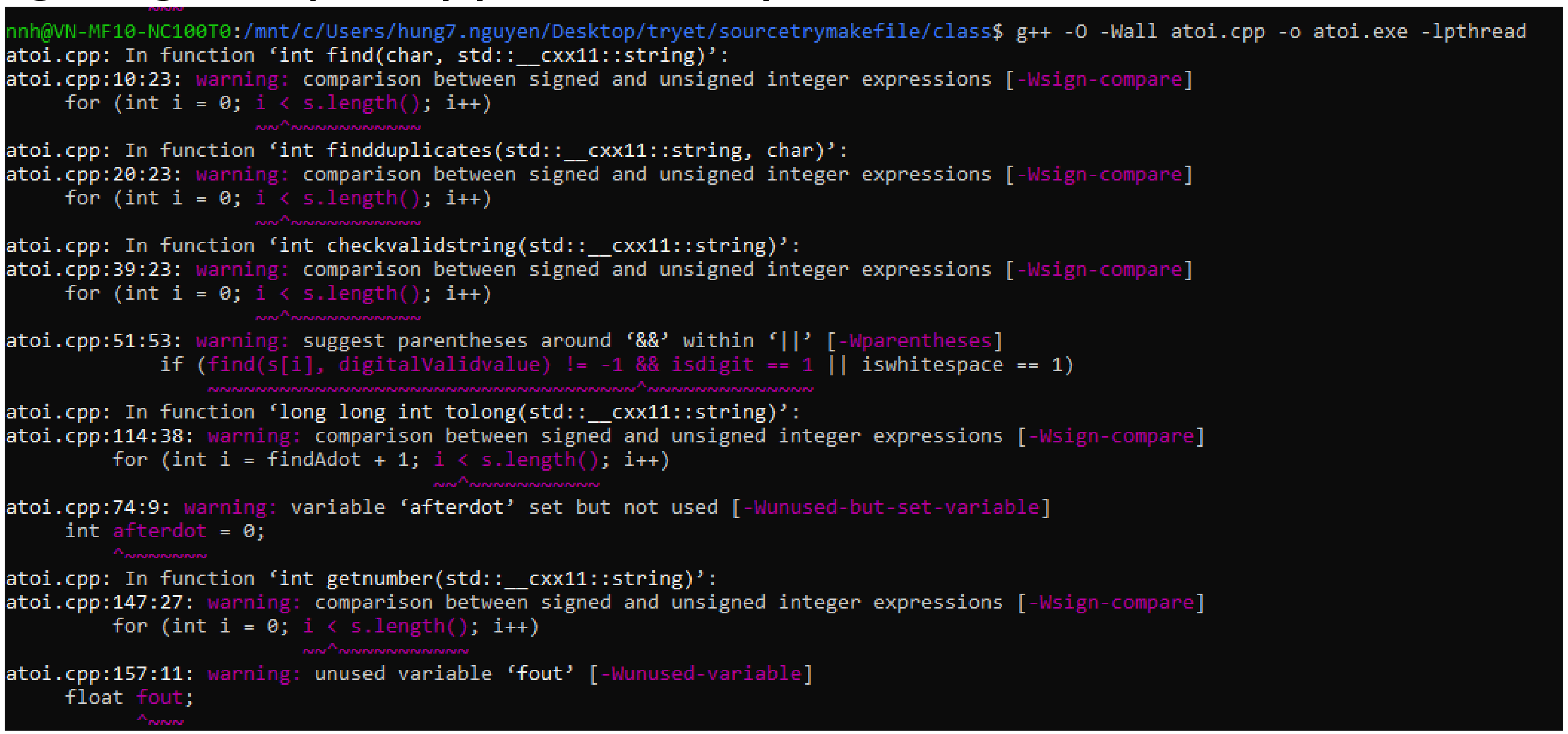
Compiler by GNU Compiler Collection

- **Some more example:**

+ Optimized code on linux: g++/gcc –O input.cpp/.c –o output file

+ Build source code have a lib example: pthreah.h

g++/gcc input.cpp/.c –o output file –l+libname without .h



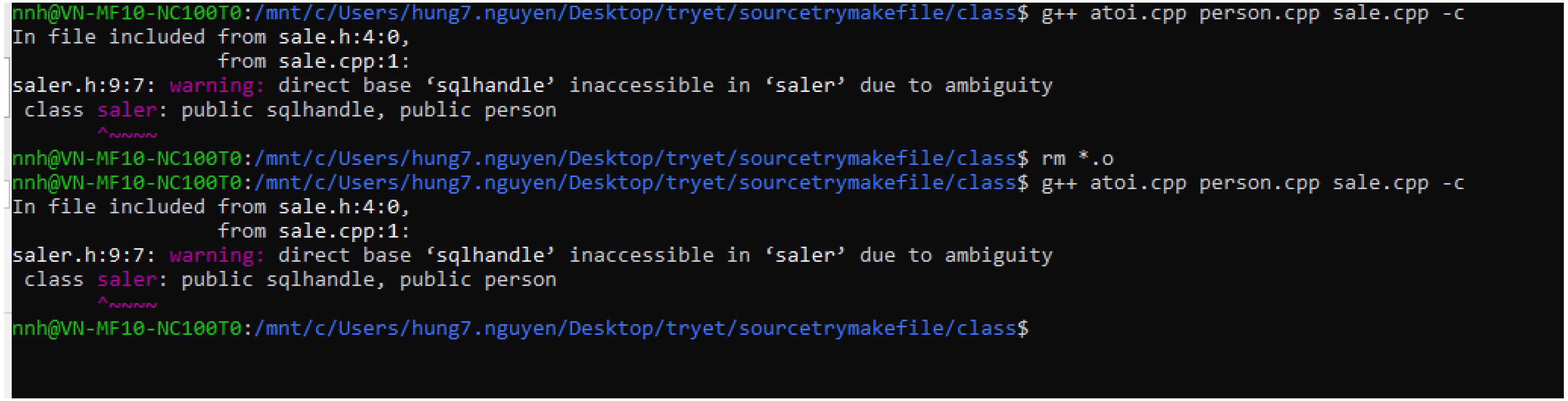
```
nnh@VN-MF10-NC100T0:/mnt/c/Users/hung7.nguyen/Desktop/tryet/sourcetrymakefile/class$ g++ -O -Wall atoi.cpp -o atoi.exe -lpthread
atoi.cpp: In function ‘int find(char, std::__cxx11::string)’:
atoi.cpp:10:23: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
     for (int i = 0; i < s.length(); i++)
                     ~~^~~~~~~~~~~~~
atoi.cpp: In function ‘int findduplicates(std::__cxx11::string, char)’:
atoi.cpp:20:23: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
     for (int i = 0; i < s.length(); i++)
                     ~~^~~~~~~~~~~~~
atoi.cpp: In function ‘int checkvalidstring(std::__cxx11::string)’:
atoi.cpp:39:23: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
     for (int i = 0; i < s.length(); i++)
                     ~~^~~~~~~~~~~~~
atoi.cpp:51:53: warning: suggest parentheses around ‘&&’ within ‘||’ [-Wparentheses]
             if (find(s[i], digitalValidvalue) != -1 && isdigit == 1 || iswhitespace == 1)
                 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^~~~~~~~~~~~~~~
atoi.cpp: In function ‘long long int tolong(std::__cxx11::string)’:
atoi.cpp:114:38: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
         for (int i = findAdot + 1; i < s.length(); i++)
                                    ~~^~~~~~~~~~~~~
atoi.cpp:74:9: warning: variable ‘afterdot’ set but not used [-Wunused-but-set-variable]
     int afterdot = 0;
         ^~~~~~~~
atoi.cpp: In function ‘int getnumber(std::__cxx11::string)’:
atoi.cpp:147:27: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
         for (int i = 0; i < s.length(); i++)
                         ~~^~~~~~~~~~~~~
atoi.cpp:157:11: warning: unused variable ‘fout’ [-Wunused-variable]
     float fout;
           ^~~~
```

Compiler by GNU Compiler Collection

- **Some more example:**

+ Multiple source files:

g++/gcc source1.cpp source2.cpp …. sourceN.cpp -o output



```
nnh@VN-MF10-NC100T0:/mnt/c/Users/hung7.nguyen/Desktop/tryet/sourcetrymakefile/class$ g++ atoi.cpp person.cpp sale.cpp -c
In file included from sale.h:4:0,
                 from sale.cpp:1:
saler.h:9:7: warning: direct base ‘sqlhandle’ inaccessible in ‘saler’ due to ambiguity
 class saler: public sqlhandle, public person
       ^~~~~
nnh@VN-MF10-NC100T0:/mnt/c/Users/hung7.nguyen/Desktop/tryet/sourcetrymakefile/class$ rm *.o
nnh@VN-MF10-NC100T0:/mnt/c/Users/hung7.nguyen/Desktop/tryet/sourcetrymakefile/class$ g++ atoi.cpp person.cpp sale.cpp -c
In file included from sale.h:4:0,
                 from sale.cpp:1:
saler.h:9:7: warning: direct base ‘sqlhandle’ inaccessible in ‘saler’ due to ambiguity
 class saler: public sqlhandle, public person
       ^~~~~
nnh@VN-MF10-NC100T0:/mnt/c/Users/hung7.nguyen/Desktop/tryet/sourcetrymakefile/class$
```

Compiler by GNU Compiler Collection

- **The difference of g++ and gcc:**

g++	gcc
g++ is used to compile C++ program.	gcc is used to compile C program.
g++ can compile any .c or .cpp files but they will be treated as C++ files only.	gcc can compile any .c or .cpp files but they will be treated as C and C++ respectively.
Command to compile C++ program through g++ is g++ fileName.cpp -o binary	command to compile C program through gcc is g++ fileName.c -o binary
Using g++ to link the object files, files automatically links in the std C++ libraries.	gcc does not do this.
g++ compiles with more predefined macros.	gcc compiles C++ files with more number of predefined macros. Some of them are #define __GXX_WEAK__ 1, #define __cplusplus 1, #define __DEPRECATED 1, etc

What is the makefile?

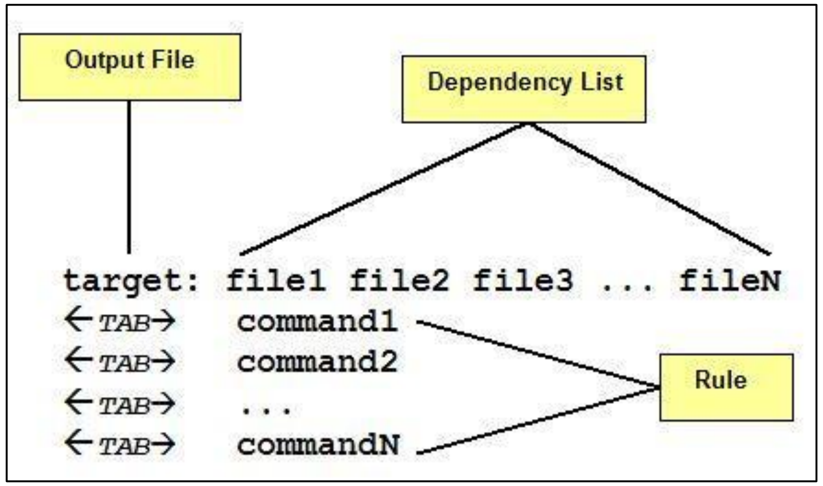
- **Intruduce Makefile:**

Makefile is a file with default name “Makefile”, containing a set of directives used by a make build automation tool generate a goal file(2).

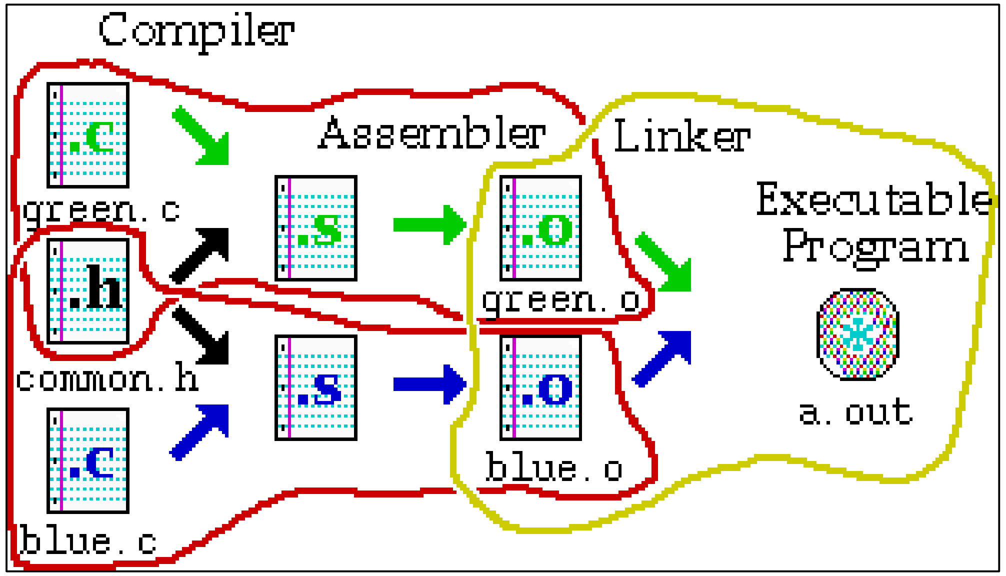
- **Structure of makefile:**

A simple makfile consists of rules with follow shape:

target … :   prerequisites …
                recipe ...

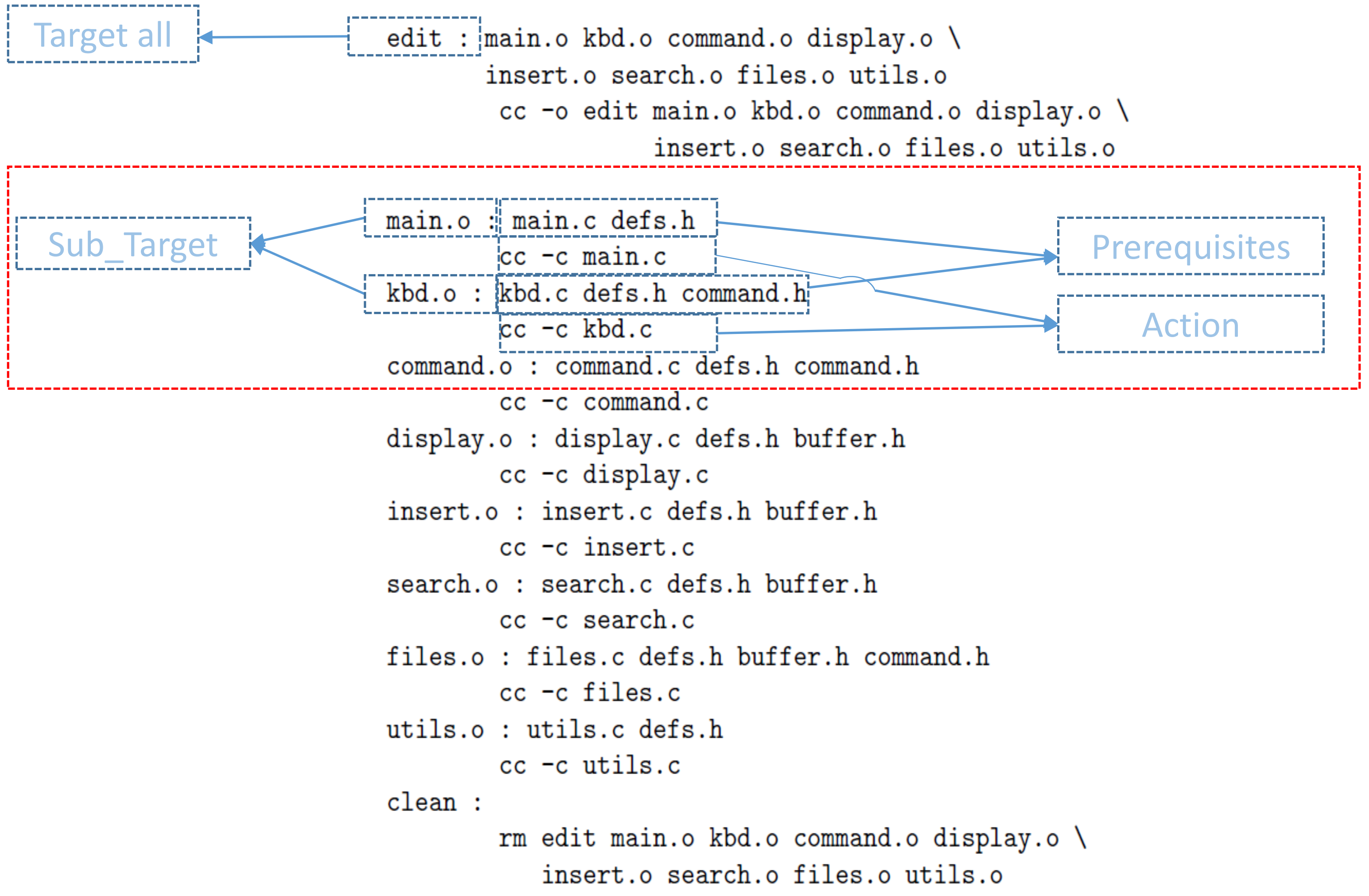


```
target: file1 file2 file3 ... fileN
←TAB→   command1
←TAB→   command2
←TAB→   ...
←TAB→   commandN
```



What is the makefile?

- **How does it work?**



```
edit : main.o kbd.o command.o display.o \
       insert.o search.o files.o utils.o
        cc -o edit main.o kbd.o command.o display.o \
                   insert.o search.o files.o utils.o

main.o : main.c defs.h
        cc -c main.c
kbd.o : kbd.c defs.h command.h
        cc -c kbd.c
command.o : command.c defs.h command.h
        cc -c command.c
display.o : display.c defs.h buffer.h
        cc -c display.c
insert.o : insert.c defs.h buffer.h
        cc -c insert.c
search.o : search.c defs.h buffer.h
        cc -c search.c
files.o : files.c defs.h buffer.h command.h
        cc -c files.c
utils.o : utils.c defs.h
        cc -c utils.c
clean :
        rm edit main.o kbd.o command.o display.o \
           insert.o search.o files.o utils.o
```

Each Sub_target is a prerequisite of target all.

=> Make will do all subtarget before make for target all, but first it will check what is target all need?

Using makefile

- **Naming:**
  - ➢makefile or Makefile are standard.
  - ➢Other name can be also used.
- **Running makefile:**
  - ➢ make with the standard name
  - ➢ make –f filename with the other name which is not “makefile or Makefile”
  - ➢ make tagert_name: if you don’t want the make file return first line target.

5 kind of things with makefile

Makefile contain 5 kind of things:

- **Explicit rule**
- **Implicit rule**
- **Variable definition (Macros)**
- **Directive (Conditional)**
- **Comment (#)**

- Additional sign “\” can help you separate a command to two row.

5 kind of things with makefile

- **Implicit rule**

➢ Implicit rules are standard ways for making one type of file from another type.

➢ There are numerous rules for making an ***.o*** file – from a ***.c*** file, a ***.p*** file, etc. `make` applies the first rule it meets.

➢ If you have not defined a rule for a given object file, `make` will apply an implicit rule for it.

5 kind of things with makefile

- **Variable in makefile**

➢By using sign “=” can a variable can vary a file name/ directory…

<u>Defining variables on the command line:</u>

Take precedence over variables defined in the makefile.

```
make C=cc
```

5 kind of things with makefile

- **Automatic variable**

```
%.o: %.cpp1 c.pp2
    $(C) -c $^ $< : g++ - c %.cpp ~ g++ -c sale.cpp %.cpp1
#rule make goal file
%: %.o $(obj)
    $(C) -o  -$@ $^
```

- ➢ `$@` - The name of the target of the rule (`sale.o`).
- ➢ `$<` - The name of the first dependency (`sale.cpp`).
- ➢ `$^` - The names of all the dependencies (`sale.cpp sale.h`).
- ➢ `$?` - The names of all dependencies that are newer than the target

5 kind of things with makefile

- **Make option**

  `-f` *filename* – when the makefile name is not standard

  `-t` – (touch) mark the targets as up to date

  `-q` – (question) are the targets up to date, exits with 0 if true

  `-n` – print the commands to execute but do not execute them

  / `-t`, `-q`, and `-n`, cannot be used together /

  `-s` – silent mode

  `-k` – keep going – compile all the prerequisites even if not able to link them.

Reference link: https://www.gnu.org/software/make/manual/html_node/Options-Summary.html

5 kind of things with makefile

- **Python target**

  - There are no prerequisites.

  - example:

```
        .PHONY : clean
        clean:
                rm $(obj)
```

  or:

```
        clean:
                rm $(obj)
```

5 kind of things with makefile

- **VPATH**

➢ Defines directory which to be search if a file not found at current folder.

```
VPATH= dir : dir
ex: VPATH = srcs: ../class/
```

➢ Using lower case -> more selective directory search:

```
ex: /vpath %.h class/
```

- Using GPATH to store the target at same location with prerequisites

- **đường dẫn tương đối.**

5 kind of things with makefile

- **example**

Explicit	Implicit
see below	see below

Explicit

```
all: saler.o main.o sqlhandle.o mystring.o sale.o
     g++  main.o sqlhandle.o mystring.o sale.o sale
r.o -o main -lsqlite3
sqlhandle.o:
    g++ -c ../class/sqlhandle.cpp
saler.o:
     g++  -c ../class/saler.cpp
sale.o:
     g++  -c ../class/sale.cpp
main.o:
     g++  -c ../main/main.cpp
mystring.o:
    g++  -c ../commonlib/mystring.cpp
clean:
    rm *.o
```

Implicit

```
CC=gcc
C=g++
CFLAGS = -c -g -Wall
obj = $(class)saler.o $(class)main.o $(class)sqlhandle.o $(commonlib)mystring.o $(
class)sale.o
hdrs = $(class)saler.h $(class)sqlhandle.h $(commonlib)mystring.h $(class)sale.h
obj/%.o: %.cpp
   $(C) -c $^
#rule make goal file
%: %.o $(obj)
   $(C) -o  -$@ $^
main : ./makefile
sqlhandle.o: sqlhandle.cpp
saler.o: saler.cpp
sale.o: sale.cpp
main.o:main.cpp
mystring.o: mystring.cpp`
clean:
   rm main $(obj)
```

C-make and examples

THANK YOU